# PR #51255 完整报告

vllm-project/vllm

[Model] Add native Dots3 NOTE multimodal support

合并时间：2026-08-12 23:13

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51255

## 执行摘要

- 一句话：为 vLLM 新增 Dots3 NOTE 全模态模型原生支持
- 推荐动作：值得精读。该 PR 展示了三类可复用的设计模式：一是混合 MLA 模型如何用“均匀填充物理 cache 行宽 + 模型私有稀疏 MLA 后端”复用现有混合 KV cache 分配器；二是显式 model-selected prefill backend class 让模型私有 FA3 实现无需模型名判断即可接入；三是 CPU 规划 /GPU 重建的双层序列长度处理规避未分配页读取。建议重点核对 mla_attention.py 共享改动对既有 MLA 模型的回归，并跟进 FIPS sha1 与模型级 CI 测试两个遗留项。

## 功能与动机

PR body 明确要求为完整 Dots3 NOTE 模型提供 vLLM 原生支持，包括文本、图像、音频、原生视频、FP8 MoE、MTP 与工具调用。Dots3 NOTE 的 DSA 层在结构上与 DeepSeek-V3.2 相关，但无法直接复用完整 DeepSeek 实现，因为它组合了带 DSA 索引的全注意力 MLA、不同注意力几何的稠密滑动窗口 MLA、两种不同的 attention gate、跨层组不同的 latent KV 宽度、尾部 RoPE 的 DSA index-key checkpoint 布局、SWA 形状的 MTP 层，以及 MoE 视觉编码器和音频编码器。PR 因此采用一个统一 Hugging Face 架构（model_type: dots3_note，architectures: Dots3NoteForCausalLM），同一架构既可服务完整多模态模型，也可通过现有 --language-model-only 选项仅运行语言模型。

## 实现拆解

整体实现按 5 步拆解：

1. 模型骨架与注册接入：新增 vllm/transformers_utils/configs/dots3_note.py 定义 Dots3NoteConfig，其中 n_group、topk_group 默认设为 1，避免继承 DeepSeek-V3 分组路由默认值而改变 expert 选择；在 vllm/model_executor/models/registry.py 注册 dots3_note 模型类型，vllm/models/dots3_note/ 按 common/nvidia 分层组织。
2. 语言模型核心与混合 MLA 注意力：nvidia/model.py 中 Dots3NoteFullAttention 复用 DeepSeek 稀疏 MLA（DSA top-k 索引），通过 Dots3NotePaddedMLAAttention 把物理 cache 行宽统一为 SWA 行宽（1088），让现有混合 KV cache 分配器同时管理 DSA（576）与 SWA 两组；_forward_note_mla 实现 NOTE 特有的 RoPE-only index-key 归一化与 headwise/ 整体两种 attention gate；Dots3NoteMoE 在 TP 世界大小与 FP8 block size 联合约束下对共享 expert 和稠密 MLP 做填充（_padded_mlp_size），并保持原始 reduction 语义。共享层改动 mla_attention.py 支持按注意力层推导 latent 维度、可选滑

动窗口（输出 SlidingWindowMLASpec）与显式 prefill backend class。

3. 滑动窗口注意力执行后端：nvidia/attention.py 内置 Triton _gather_swa_kv_kernel 与 _apply_swa_score_mask_kernel；prefill 与混合 batch 走 FA3 varlen MHA，并把 V 从 128 padding 到 256 以规避 FA3 在 Hopper 上不支持 Q/K=256、V=128 不同 head dim 的限制；decode-only batch 走 absorbed latent MQA。
_build_sliding_window_metadata 用 CPU 序列长度做 chunk 规划、在 GPU 上重建精确的 KV spans、token-to-request 映射与 cu_seq_lens，避免 gather 读取未分配 block。
4. 多模态塔与视频链路：vision.py/vision_attention.py 实现 MoE 视觉 Transformer（金字塔路由、fp32 路由计算保持数值稳定、FP8 MoE 融合内核）；
audio.py/audio_encoder.py 移植 Whisper 系 Dots 语音编码器并做分块 mel 谱预处理；
multimodal.py 的 Dots3NoteForCausalLM 统一组装 embed_multimodal，视频经 video.py 的 _solve_degrade 在视觉 token 预算下联合降级 fps 与 patch 数，并抽取音轨与帧按时间戳交错。
5. MTP、工具解析与测试配套：mtp.py 注册 Dots3NoteMTP，复用 DeepSeek-V3.2 MTP 编排，仅适配 checkpoint 命名、SWA 几何与 dense MLP padding（_adapt_weights）；新增 vllm/tool_parsers/dots_tool_parser.py 提供 Dots XML 解析器（多 invoke、schema 感知参数类型转换、JSON fallback），默认可通过 --tool-call-parser dots 选用；测试配套为 tests/tool_parsers/test_dots_tool_parser.py，覆盖注册、多 invoke、类型转换与 JSON fallback，模型侧则以 8 卡 Hopper FP8 checkpoint 做手工 smoke 验证。

关键文件：

- vllm/models/dots3_note/nvidia/model.py（模块 模型核心；类别 source；类型 core-logic；符号 _padded_mlp_size, Dots3NoteMoE, _forward_note_mla, Dots3NotePaddedMLAAttention）：Dots3 NOTE 语言模型主体：复用 DeepSeek MLA/MoE 组件，新增 NOTE 特有的 RoPE-only index-key 归一化、headwise/ 整体 attention gate、FP8 共享 expert 填充与物理 cache 行宽统一，是整个 PR 的核心。
- vllm/models/dots3_note/nvidia/attention.py（模块 注意力后端；类别 source；类型 core-logic；符号 _gather_swa_kv_kernel, _apply_swa_score_mask_kernel, Dots3NoteDecodeMetadata, _SlidingWindowChunk）：滑动窗口 MLA 的模型私有执行后端：Triton gather/ 掩码内核、FA3 varlen prefill（V padding 规避不同 head dim 限制）与 absorbed MQA decode，CPU 规划 GPU 重建是正确性关键。
- vllm/models/dots3_note/common/processor.py（模块 多模态处理；类别 source；类型 data-contract；符号 load_note_config_section, Dots3NoteImageProcessor, resized_size, preprocess）：多模态数据处理契约：镜像 Cybertron ViT 的 CPU 图像预处理、prompt 占位符组装（img/audio/video pad token）、token 预算与 dummy inputs，是多模态请求进入模型前的关键边界。
- vllm/models/dots3_note/common/video.py（模块 视频预处理；类别 source；类型 core-logic；符号 Dots3NoteVideoPart, _compute_target_size, _real_patches_at, _frame_hard_cap）：训练一致的原生视频预处理：解码、视觉 token 预算求解、音轨抽取与帧 / 音频交错，直接决定长视频场景的 token 成本与质量权衡。
- vllm/models/dots3_note/nvidia/multimodal.py（模块 多模态组装；类别 source；类型 data-contract；符号 Dots3NoteForCausalLM, get_placeholder_str,

_process_image_input, _process_audio_input）：统一 Omni 架构的组装层：视觉 / 音频塔可选加载，按帧序与音频序把多模态 embedding 交错成单个替换 span，并处理视频的 image/audio 消费一致性校验。

- vllm/models/dots3_note/nvidia/audio.py（模块 音频编码；类别 source；类型 core-logic；符号 Dots3NoteAudioConfig, DotsEncoderWithMask, encode_waveform, log_mel_spectrogram）：NOTE 音频配置与波形预处理：分块 mel 谱、token 长度推导与 Dots 语音编码器封装，音频 token 预算直接决定 prompt 占位符数量。
- vllm/models/dots3_note/nvidia/audio_encoder.py（模块 音频编码；类别 source；类型 core-logic；符号 RMSNorm, RotaryEmbedding, get_cos_sin, apply_rotary_pos_emb）：从 cybertron_alm 移植的 Dots 语音编码器（Whisper 系），支持 FA3 varlen 与 eager 双路径，是音频理解能力的核心网络。
- vllm/models/dots3_note/nvidia/vision.py（模块 视觉编码；类别 source；类型 core-logic；符号 DotsMoEVitConfig, MoESwiGLUFFN, DotsSwiGLUFFN, RMSNorm）：MoE 视觉 Transformer：金字塔路由、fp32 路由保持数值稳定、FP8 MoE 融合内核，视觉 token 是视频预处理器与图像理解的基础。
- vllm/model_executor/layers/attention/mla_attention.py（模块 MLA 层；类别 source；类型 data-contract；符号 MLAAttention, SlidingWindowMLASpec）：共享改动：MLA 元数据构建改为从每个 KV-cache group 关联的注意力层推导 latent 维度，并支持滑动窗口缓存规格与显式 prefill backend class，是所有 MLA 模型的公共回归面。
- vllm/tool_parsers/dots_tool_parser.py（模块 工具解析；类别 source；类型 feature；符号 DotsToolParser, _extract_name, _convert_param_value, _resolve_param_type）：新增 Dots XML 工具调用解析器：多 invoke、schema 感知参数类型转换与 JSON fallback，作为可选 tool-call-parser 注册，不影响默认行为。
- vllm/transformers_utils/configs/dots3_note.py（模块 模型配置；类别 source；类型 core-logic；符号 Dots3NoteConfig）：Dots3NoteConfig 默认 n_group/topk_group=1，避免继承 DeepSeek 分组路由默认值改变 expert 选择，是模型正确性的配置地基。
- tests/tool_parsers/test_dots_tool_parser.py（模块 工具解析测试；类别 test；类型 test-coverage；符号 _tool, _request, _stream, parser）：当前 PR 中唯一成规模的自动化测试配套，覆盖 parser 注册、多 invoke、schema 类型转换与 JSON fallback，是后续模型级测试的参照模板。

关键符号：Dots3NoteForCausalLM.embed_multimodal, _process_video_input, Dots3NoteFlashAttnPrefillBackend.run_sliding_window, _build_sliding_window_metadata, _gather_swa_kv_kernel, _apply_swa_score_mask_kernel, _forward_note_mla, Dots3NotePaddedMLAAttention.get_kv_cache_spec, Dots3NoteMoE.forward, _padded_mlp_size, _solve_degrade, Dots3NoteImageProcessor.preprocess, Dots3NoteMTP._adapt_weights, DotsToolParser._parse_xml_invoke, Dots3NoteAudioModel.forward

## 关键源码片段

### vllm/models/dots3_note/nvidia/model.py

Dots3 NOTE 语言模型主体：复用 DeepSeek MLA/MoE 组件，新增 NOTE 特有的 RoPE-only index-key 归一化、headwise/ 整体 attention gate、FP8 共享 expert 填充与物理 cache 行宽统一，是整个 PR 的核心。

```
# vllm/models/dots3_note/nvidia/model.py
# NOTE 专属的 MLA 前向：相比 DeepSeek 实现，额外引入 RoPE-only index-key 归一化
# （k_rope_only_layernorm）与模型自定义 attention gate，支持 headwise 与整体缩放两种形态。
def _forward_note_mla(
    attention,
    positions: torch.Tensor,
    hidden_states: torch.Tensor,
    *,
    g_proj: nn.Module,
    k_rope_only_layernorm: nn.Module,
    attention_gate_type: str,
    q_lora_scale: float,
    kv_lora_scale: float,
    llama_4_scaling: torch.Tensor | None = None,
) -> torch.Tensor:
    qkv_lora = attention.fused_qkv_a_proj(hidden_states)[0]
    # 低秩投影输出拆成 query 侧与 KV 侧两部分：q_c 与 kv_lora
    q_c, kv_lora = qkv_lora.split(
        [attention.q_lora_rank, attention.kv_lora_rank + attention.qk_rope_head_dim],
        dim=-1,
    )
    q_c = attention.q_a_layernorm(q_c) * q_lora_scale
    kv_c, k_pe = kv_lora.split(
        [attention.kv_lora_rank, attention.qk_rope_head_dim], dim=-1
    )
    kv_c_normed = attention.kv_a_layernorm(kv_c) * kv_lora_scale
    # NOTE 对 RoPE-only 的 index key 单独做 RMSNorm，这是与 DeepSeek 的关键差异之一
    k_pe = k_rope_only_layernorm(k_pe).unsqueeze(1)

    q = attention.q_b_proj(q_c)[0]
    heads = attention.num_heads
    if attention.dcp_q_replicate:
        heads *= attention.q_b_proj.group_size
    q = q.view(-1, heads, attention.qk_head_dim)
    q[..., attention.qk_nope_head_dim :], k_pe = attention.rotary_emb(
        positions, q[..., attention.qk_nope_head_dim :], k_pe
    )

    # DSA 全注意力层走 top-k 索引器，稀疏 MLA 路径复用 DeepSeek 的索引器实现
    if attention.indexer and attention.is_sparse and not attention.skip_topk:
        attention.indexer(hidden_states, q_c, positions, attention.indexer_rope_emb)
    if llama_4_scaling is not None:
        q *= llama_4_scaling

    q_dcp_replicated = None
```

```
    if attention.dcp_q_replicate:
        q_dcp_replicated, q = q, attention.q_b_proj._local_view(q)
    attn_out = attention.mla_attn(
        q,
        kv_c_normed,
        k_pe,
        output_shape=(
            hidden_states.shape[0],
            attention.num_heads * attention.v_head_dim,
        ),
        q_dcp_replicated=q_dcp_replicated,
    )

    # 模型自定义 attention gate：headwise 时每个 head 一个标量权重，
    # 需要按 TP rank 切出本 rank 的 head 段；整体模式则直接逐 token 缩放
    gate = g_proj(hidden_states)[0]
    if attention_gate_type == "headwise":
        if gate.shape[-1] != attention.num_heads:
            rank = get_tensor_model_parallel_rank()
            gate = gate.narrow(-1, rank * attention.num_heads, attention.num_heads)
        attn_out = attn_out.view(-1, attention.num_heads, attention.v_head_dim)
        gate = torch.sigmoid(gate.float()).to(attn_out.dtype)
        attn_out = (attn_out * gate.unsqueeze(-1)).flatten(-2)
    else:
        gate = torch.sigmoid(gate.float()).to(attn_out.dtype)
        attn_out = attn_out * gate
    return attention.o_proj(attn_out)[0]
```

## vllm/models/dots3_note/nvidia/attention.py

滑动窗口 MLA 的模型私有执行后端：Triton gather/ 掩码内核、FA3 varlen prefill（V padding 规避不同 head dim 限制）与 absorbed MQA decode，CPU 规划 GPU 重建是正确性关键。

```
# vllm/models/dots3_note/nvidia/attention.py
# 滑动窗口 MLA 的 gather 规划：先在 CPU 上按 workspace 容量把请求切成多个 chunk，
# 每个 chunk 单独构造 FA3 varlen 所需的 cu_seq_lens 与 token_to_seq 映射；
# 只在 GPU 上重建张量，避免做请求级的大 gather。
def _build_sliding_window_metadata(
    *,
    seq_lens_cpu: torch.Tensor,
    query_start_loc_cpu: torch.Tensor,
    sliding_window: int,
    workspace: torch.Tensor,
    workspace_size: int,
    device: torch.device,
) -> _SlidingWindowMetadata:
    # 用 CPU 端长度做规划：只负责分块，不触碰潜在未分配的物理页
    query_lens_cpu = (query_start_loc_cpu[1:] - query_start_loc_cpu[:-1]).to(
        dtype=torch.int32
```

```
)
seq_lens_cpu = seq_lens_cpu.to(dtype=torch.int32)
# 每个请求实际需要的 KV 长度受滑动窗口约束，而不是完整历史长度
kv_lens_cpu = torch.minimum(seq_lens_cpu, query_lens_cpu + sliding_window - 1)
starts_cpu = seq_lens_cpu - kv_lens_cpu

chunks: list[_SlidingWindowChunk] = []
req_start = 0
while req_start < query_lens_cpu.numel():
    req_end = req_start
    num_kv_tokens = 0
    # 贪心装填：把多个请求塞进同一个 chunk，直到超出 MLA workspace 容量
    while req_end < query_lens_cpu.numel():
        next_len = int(kv_lens_cpu[req_end].item())
        if num_kv_tokens and num_kv_tokens + next_len > workspace_size:
            break
        if next_len > workspace_size:
            raise ValueError(
                "Dots3 NOTE SWA prefill window exceeds the MLA workspace: "
                f"{next_len} > {workspace_size}"
            )
        num_kv_tokens += next_len
        req_end += 1

    query_lens = query_lens_cpu[req_start:req_end]
    kv_lens = kv_lens_cpu[req_start:req_end]
    num_reqs = req_end - req_start
    cu_seq_lens_q_cpu = torch.zeros(num_reqs + 1, dtype=torch.int32)
    cu_seq_lens_k_cpu = torch.zeros(num_reqs + 1, dtype=torch.int32)
    torch.cumsum(query_lens, 0, out=cu_seq_lens_q_cpu[1:])
    torch.cumsum(kv_lens, 0, out=cu_seq_lens_k_cpu[1:])
    # 把 flatten 后的 KV token 映射回所属请求，FA3 varlen 需要这个映射
    token_to_seq_cpu = torch.repeat_interleave(
        torch.arange(num_reqs, dtype=torch.int32), kv_lens
    )
    query_start = int(query_start_loc_cpu[req_start].item())
    query_end = int(query_start_loc_cpu[req_end].item())
    chunks.append(
        _SlidingWindowChunk(
            req_start=req_start,
            req_end=req_end,
            query_start=query_start,
            query_end=query_end,
            cu_seq_lens_q=cu_seq_lens_q_cpu.to(device, non_blocking=True),
            cu_seq_lens_k=cu_seq_lens_k_cpu.to(device, non_blocking=True),
            starts=starts_cpu[req_start:req_end].to(device, non_blocking=True),
            token_to_seq=token_to_seq_cpu.to(device, non_blocking=True),
            num_kv_tokens=num_kv_tokens,
            max_seq_len_q=int(query_lens.max().item()),
```

```
                max_seq_len_k=int(kv_lens.max().item()),
            )
        )
        req_start = req_end

    return _SlidingWindowMetadata(chunks=chunks, workspace=workspace)
```

## vllm/models/dots3_note/common/video.py

训练一致的原生视频预处理：解码、视觉 token 预算求解、音轨抽取与帧 / 音频交错，直接决定长视频场景的 token 成本与质量权衡。

```
# vllm/models/dots3_note/common/video.py
# 视频预处理的核心预算求解器：在视觉 token 预算约束下，
# 同时决定抽帧数量（fps）与每帧 patch 上限（patch_cap）。
# 优先保持完整质量；超预算时用二分搜索在 fps 与 patch 之间做联合降级。
def _solve_degrade(
    visual_budget: int,
    duration: float,
    orig_h: int,
    orig_w: int,
    orig_fps: float,
    seq_length: int,
) -> tuple[int, int]:
    aligned_h = max(_ALIGN, round(orig_h / _ALIGN) * _ALIGN)
    aligned_w = max(_ALIGN, round(orig_w / _ALIGN) * _ALIGN)
    orig_max_pf = (aligned_h // _ALIGN) * (aligned_w // _ALIGN)
    fps_cap = min(_FPS_CAP, max(orig_fps, 1e-6))
    pf_cap = min(_PF_CEIL, max(orig_max_pf, _PF_FLOOR))
    frame_cap = _frame_hard_cap(seq_length)

    # scale=1.0 表示最高质量：最高 fps 与最大 patch 数；scale=0.0 为最低质量
    def usage(scale: float) -> tuple[int, float, int, int]:
        fps = _FPS_MIN + scale * (fps_cap - _FPS_MIN)
        patch_cap = _PF_FLOOR + scale * (pf_cap - _PF_FLOOR)
        num_frames = max(_MIN_FRAMES, min(int(round(duration * fps)), frame_cap))
        patches = _real_patches_at(orig_h, orig_w, int(round(patch_cap)))
        # 每帧成本 = 实际 patch 数 + 固定帧开销（如时间戳等额外 token）
        return (
            num_frames * (patches + _FRAME_OVERHEAD),
            fps,
            int(round(patch_cap)),
            num_frames,
        )

    # 最高质量仍在预算内：直接返回
    if usage(1.0)[0] <= visual_budget:
        _, _, patch_cap, num_frames = usage(1.0)
        return num_frames, patch_cap
```

```
    # 最低质量都超预算：退化为尽可能多塞帧，每帧取最小 patch 数
    floor_cost = _real_patches_at(orig_h, orig_w, _PF_FLOOR) + _FRAME_OVERHEAD
    if usage(0.0)[0] > visual_budget:
        return max(_MIN_FRAMES, min(visual_budget // floor_cost, frame_cap)), _PF_FLOOR

    # 预算落在两个极端之间：二分查找最大可行 scale（单调递减）
    low, high = 0.0, 1.0
    for _ in range(50):
        mid = (low + high) / 2
        if usage(mid)[0] <= visual_budget:
            low = mid
        else:
            high = mid
    _, _, patch_cap, num_frames = usage(low)
    return num_frames, patch_cap
```

# 评论区精华

核心 review 交锋如下：

- zyongye 质疑死代码：在 vllm/v1/attention/backends/mla/flashattn_mla_sparse.py 询问 "Can we remove these lines since it's not used?"，针对的是“先 flatten 完整 MLA entry 再切分 K/V”改动后遗留的旧切分逻辑。PR 最终描述明确“不再修改通用 sparse-MLA flatten 路径、通用 Triton decode 内核与 KV-cache zeroer”，说明相关共享改动在后续提交中收敛或撤销。
- depthfirst-app[bot] 安全提示（MEDIUM）：vllm/models/dots3_note/common/video.py 中线 471-473 的三处 hashlib.sha1() 调用缺少 usedforsecurity=False，在 FIPS 主机上会抛 ValueError 导致服务崩溃；仓库惯例是 vllm/utils/hashing.py 的 safe_hash() 显式关闭 FIPS 限制。该评论无人工回复，合并时未确认是否修复。
- DarkLight1337 要求更新 PR 描述：正文已包含多模态部分，与 "Current Scope" 小节矛盾，作者更新后一致。
- jeejeelee 的合并决策：最终 APPROVED 并明确 "Let's land this PR first, and continue optimizing in follow-up PRs."——接受大而全的首版实现，遗留优化放到后续 PR。
  - flashattn_mla_sparse.py 中不再使用的行是否可删除 (design): PR 最终描述明确“不再修改通用 sparse-MLA flatten 路径、通用 Triton decode 内核与 KV-cache zeroer”，说明该共享改动在后续提交中收敛或撤销；zyongye 的疑问因此闭环。
  - sha1 调用缺少 usedforsecurity=False 的 FIPS 兼容风险 (security): 无人工回复或代码修订记录，合并时未确认是否修复；建议后续补丁统一收口到 safe_hash() 或在调用处补 usedforsecurity=False。
  - 模型文件路径命名建议 (style): 最终按该路径落地，命名统一为 Dots3Note。
  - PR 描述与 Current Scope 小节不一致 (documentation): 作者更新 PR 描述，最终正文完整覆盖文本 / 图像 / 音频 / 视频全模态与工具调用。
  - 先合并再持续优化的合并决策 (other): 同意合并，遗留优化与潜在问题进入 follow-up PR。

- pre-commit 反复失败与 CI 触发 (style): 作者与维护者逐轮修复格式问题，最终所有检查通过并合并。

## 风险与影响

- 风险：风险点按文件与逻辑具体化如下：
- 共享组件回归面：vllm/model_executor/layers/attention/mla_attention.py 的 per-layer latent 维度推导改动影响所有 MLA 模型（DeepSeek 系列、Kimi、Qwen 等），任何维度推导偏差都会破坏既有模型的 KV cache 内存规划；zyongye 的评论表明该文件曾被改动后收敛，回归验证依赖历史 ML 模型测试。
- 混合 batch 正确性：_build_sliding_window_metadata 先以 CPU 序列长度规划、再在 GPU 重建索引，PR body 特别强调"防止 gather 读取未分配或未填充的 block"，说明这是已踩过的坑；若 CPU/GPU 长度不同步仍可能产生非法内存访问。
- FA3 版本与硬件兼容：Dots3NoteFlashAttnPrefillBackend 依赖 FA3 在 SM90 上的 V-padding 技巧，FA3 行为变化即失效；该后端位于 nvidia/ 目录且仅 Hopper 验证，虽然 PR 标签含 rocm，但当前无 ROCm/ 其他架构覆盖。
- FIPS 安全合规：common/video.py 三处 sha1() 未加 usedforsecurity=False，FIPS 主机上直接崩溃，且评论未闭环。
- 测试覆盖薄弱：文件列表中模型级自动化测试缺失，仅 tool parser 有单元测试；多模态与注意力路径依赖手工 smoke 验证，CI 回归能力有限。
- 性能风险：音频编码 encode_waveform 按 chunk 串行 eager 计算，视频预处理每请求重新解码并做 JPEG 往返，长视频场景 CPU 开销与首 token 延迟可能偏高。
- 影响：影响范围与程度：
- 用户：可直接用 dots3_note 模型类型服务 Dots3 NOTE 的文本 / 图像 / 音频 / 视频请求，启用 FP8、MTP 与 --tool-call-parser dots；--language-model-only 下可跳过视觉 / 音频塔加载。
- 系统：混合 KV cache 分配器与 MLA 元数据构建行为发生变化，所有 MLA 模型的服务启动路径受影响；新增 workspace 用量规划逻辑。
- 团队：新增约 6.4k 行代码、21 个提交，命名经历多轮收敛（Normalize Dots3Note naming），维护者明确以 follow-up PR 继续优化的策略，后续维护负担与 review 成本都较高。
- 风险标记：大面新代码（+6468 行），共享 MLA 组件改动，模型级 CI 测试缺失，FIPS 环境 sha1 兼容风险，Hopper/SM90 专属后端，混合 batch 依赖 CPU/GPU 长度一致

## 关联脉络

- PR #51843 [Bugfix] Disable fine-grained prefix-cache hits for incompatible hybrid KV layouts: 同一混合 KV cache 分配器 / 前缀缓存区域：Dots3 NOTE 的 DSA 与 SWA 双组不同行宽 cache 依赖该 PR 修复的兼容性语义。
- PR #51831 [Model] Support R3 capture with DeepGEMM MegaMoE: 同为 DeepSeek 系模型接入与 MoE/MTP 推理演进，NOTE 的 FP8 MoE 与 Dots3NoteMTP 可复用其捕获与编排经验。

- PR #51913 [Attention] Move context_lens_tensor compute into GDN prefill path: 同属 v1 注意力后端 prefill 路径的性能演进，NOTE SWA prefill 后端与 GDN 的改动方向一致（减少冗余计算、按需重建）。
- PR #51738 [Perf] Avoid more GPU<->CPU syncs on the model execution path: NOTE 注意力与音频编码中“CPU 规划、GPU 重建、避免设备同步”的设计与该 PR 消除 GPU-CPU 同步的方向一致。
- PR #51139 [Bugfix][Multimodal] Invalidate retained PyNvVideoCodec decoder after failure: 同属多模态视频输入前端：NOTE 原生视频处理与 PyNvVideoCodec 解码链路共同构成 vLLM 视频输入能力的演进脉络。